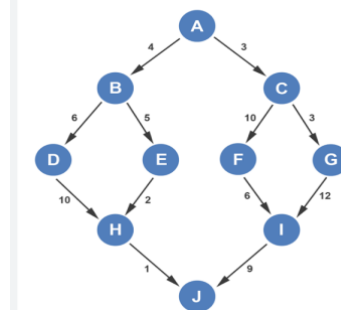


The course for terminal includes: Agents, searches (Adversarial Search, GBFS, Hill Climbing, GA, Heuristically informed Methods, Uniform Cost Search, A*), First Order Logic and Knowledge-Based Agents, Uncertainty, Planning and Constraint Satisfaction Problems.

Q1. Express the following sentences using first-order logic (FOL). Alongside universal and/or existential quantifiers, use appropriate symbols for objects and functions.

Sentences	FOL
Frank bought a dvd	$\text{Bought}(\text{Frank}, \text{dvd})$
Someone bought everything	$\exists x \forall y. \text{bought}(x, y)$
There is a student who is loved by every other student	$\exists x. (\text{Student}(x) \wedge \forall y. (\text{Student}(y) \wedge \neg(x=y) \supset \text{Loves}(x, y)))$
Every student takes at least one course.	$\forall x. (\text{Student}(x) \supset \exists y. (\text{Course}(y) \wedge \text{Takes}(x, y)))$
Not every dog is white	$\neg \forall x. \text{Dog}(x) \supset \text{White}(x)$
Bill has at most one sister.	$\forall x \forall y. (\text{SisterOf}(x, \text{Bill}) \wedge \text{SisterOf}(y, \text{Bill}) \supset x = y)$

Q2: For each of the following search strategies, give the path that would be returned from “a to j”, or write none if no path will be returned. If there are any ties, assume alphabetical tiebreaking (i.e., nodes for states earlier in the alphabet are expanded first in the case of ties).



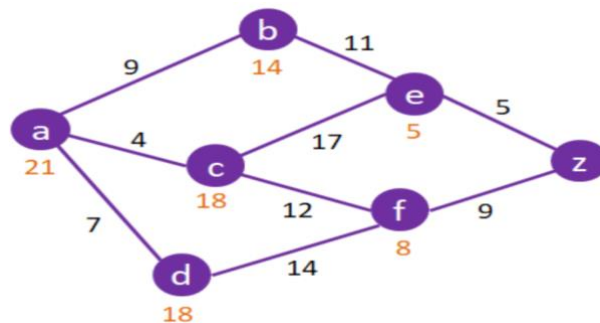
Search Type	List of States
Greedy Search algorithm (Highest cost)	A, B, D, H, J
Uniform Cost Search (Path with lowest cost)	A, C, G, I, J
A-Star Search (Path with highest cost)	A, B, D, H, J

Question 3. Express the following sentences using first-order logic (FOL). Alongside universal and/or existential quantifiers, use appropriate symbols for objects and functions.

Sentences	FOL
No students love Bill	$\neg \exists x. (\text{Student}(x) \wedge \text{Loves}(x, \text{Bill}))$
Every student takes at least one course	$\forall x. (\text{Student}(x) \supset \exists y. (\text{Course}(y) \wedge \text{Takes}(x, y)))$

Every Dog has a Tail	$\forall x. \text{Dog}(x) \supset \exists y(\text{PartOf}(x, y) \wedge \text{Tail}(y))$
Everyone bought something	$\forall x \exists y. \text{bought}(x, y)$
There exists a smart student	$\exists x. (\text{Student}(x) \wedge \text{Smart}(x))$
Bill is a student	$\text{Student}(\text{Bill})$

Q4: For each of the following search strategies, give the path that would be returned from “a to z”, or write none if no path will be returned. If there are any ties, assume alphabetical tiebreaking (i.e., nodes for states earlier in the alphabet are expanded first in the case of ties).



Search Type	List of States
Greedy Search algorithm (Highest cost)	None
Uniform Cost Search (Path with lowest cost)	A, C, F, Z
A-Star Search (Path with highest cost)	None

Iteration Techniques working:

Greedy Search Algorithm	Iterate by seeing at the heuristic cost. Its Follow DFS algorithm.
Uniform Cost Search	Iterate by seeing the Path Cost (Actual Cost). Cost sum after each iteration I -e Cumulative cost By comparing cumulative cost at point, we find optimal path
A Star	Actual Cost + Heuristic Cost So, $f(n) = g(n) + h(n)$ where $g(n)$ = cost of traversing from one node to another. This will vary from node to node $h(n)$ = heuristic approximation of the node's value. This is not a real value but an approximation cost

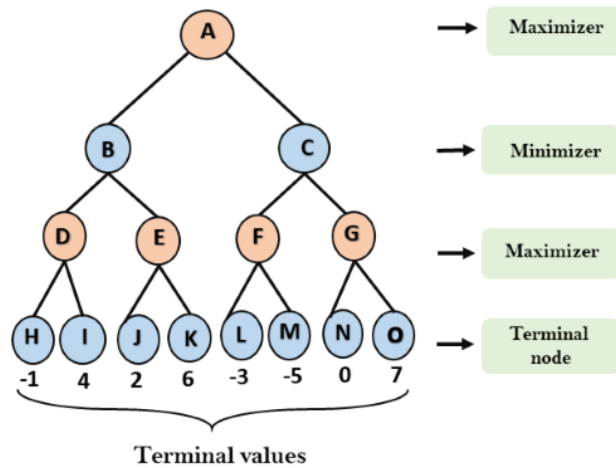
Mini-Max Algorithm in Artificial Intelligence

- i. Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- ii. Mini-Max algorithm uses recursion to search through the game-tree.
- iii. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- iv. In this algorithm two players play the game, one is called MAX and other is called MIN.
- v. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- vi. Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- vii. The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- viii. The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm:

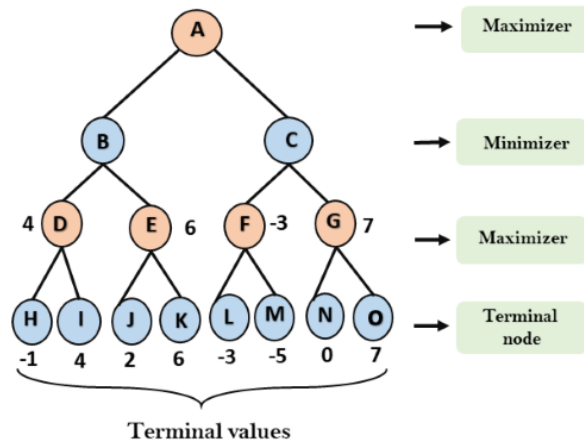
- i. The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- ii. In this example, there are two players one is called Maximizer and other is called Minimizer.
- iii. Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- iv. This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- v. At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value = - infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



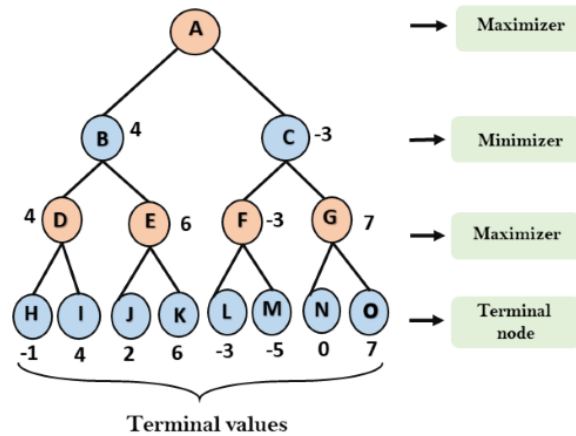
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is $-\infty$, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- i. For node D $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$
- ii. For Node E $\max(2, -\infty) \Rightarrow \max(2, 6) = 6$
- iii. For Node F $\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$
- iv. For node G $\max(0, -\infty) = \max(0, 7) = 7$



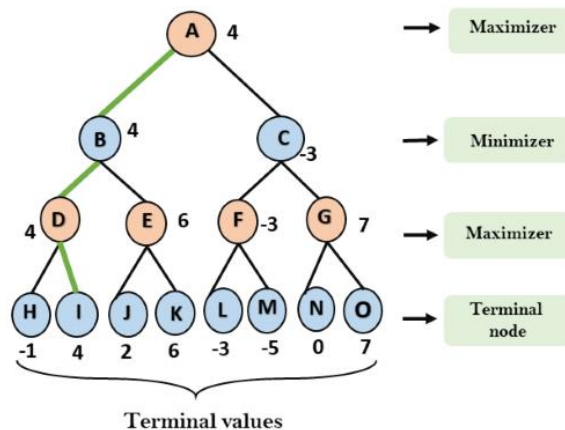
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with $+\infty$, and will find the 3rd layer node values.

- i. For node B $= \min(4, 6) = 4$
- ii. For node C $= \min(-3, 7) = -3$



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- i. For node A $\max(4, -3) = 4$



That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- i. **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- ii. **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- iii. **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(b^m)$, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- iv. **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is $O(bm)$.

Alpha-Beta Pruning

- i. Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- ii. As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- iii. Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- iv. The two-parameter can be defined as:
 1. **Alpha**: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is $-\infty$.
 2. **Beta**: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.
- v. The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

1. $\alpha \geq \beta$

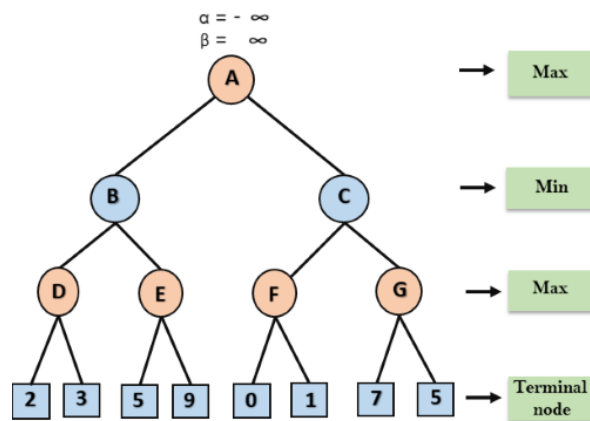
Key points about alpha-beta pruning:

- i. The Max player will only update the value of alpha.
- ii. The Min player will only update the value of beta.
- iii. While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- iv. We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

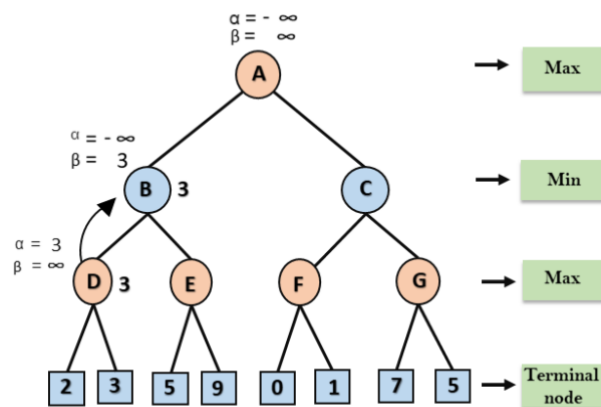
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where $\alpha = -\infty$ and $\beta = +\infty$, these value of alpha and beta passed down to node B where again $\alpha = -\infty$ and $\beta = +\infty$, and Node B passes the same value to its child D.



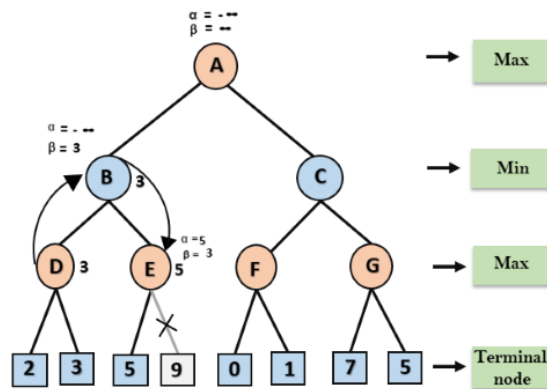
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the $\max(2, 3) = 3$ will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now $\beta = +\infty$, will compare with the available subsequent nodes value, i.e. $\min(\infty, 3) = 3$, hence at node B now $\alpha = -\infty$, and $\beta = 3$.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of $\alpha = -\infty$, and $\beta = 3$ will also be passed.

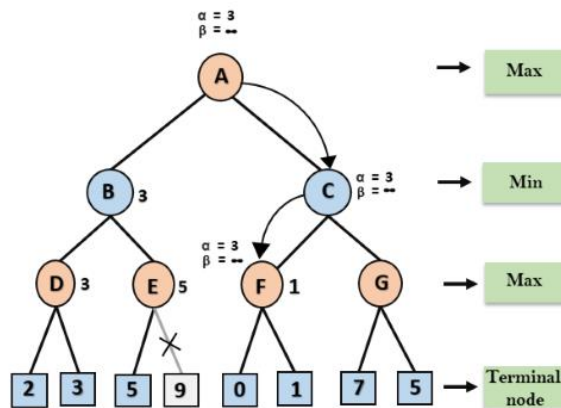
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so $\max(-\infty, 5) = 5$, hence at node E $\alpha = 5$ and $\beta = 3$, where $\alpha > \beta$, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



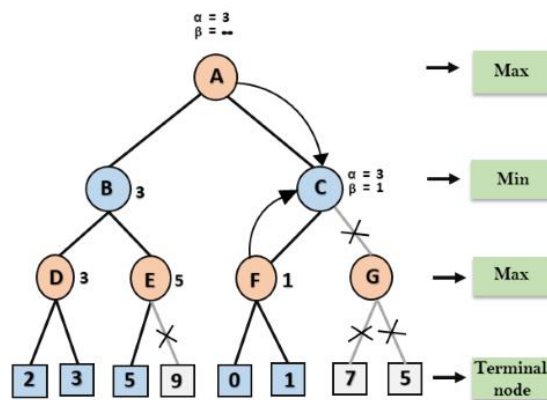
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as $\max(-\infty, 3) = 3$, and $\beta = +\infty$, these two values now passes to right successor of A which is Node C.

At node C, $\alpha=3$ and $\beta = +\infty$, and the same values will be passed on to node F.

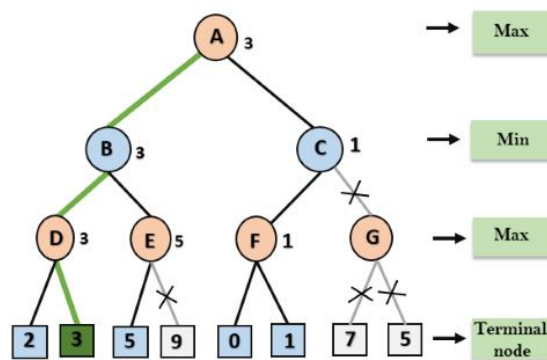
Step 6: At node F, again the value of α will be compared with left child which is 0, and $\max(3,0)=3$, and then compared with right child which is 1, and $\max(3,1)=3$ still α remains 3, but the node value of F will become 1.



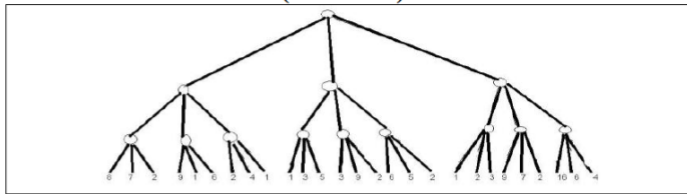
Step 7: Node F returns the node value 1 to node C, at C $\alpha=3$ and $\beta = +\infty$, here the value of beta will be changed, it will compare with 1 so $\min(\infty, 1) = 1$. Now at C, $\alpha=3$ and $\beta=1$, and again it satisfies the condition $\alpha \geq \beta$, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



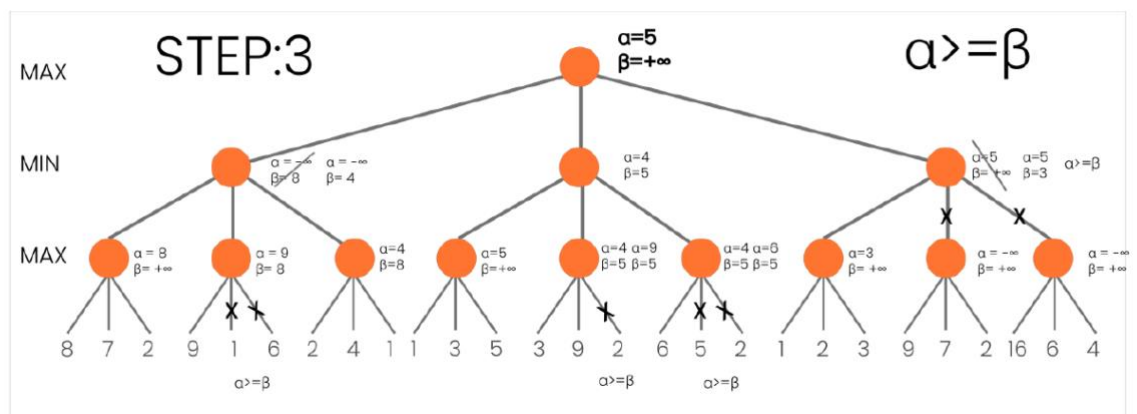
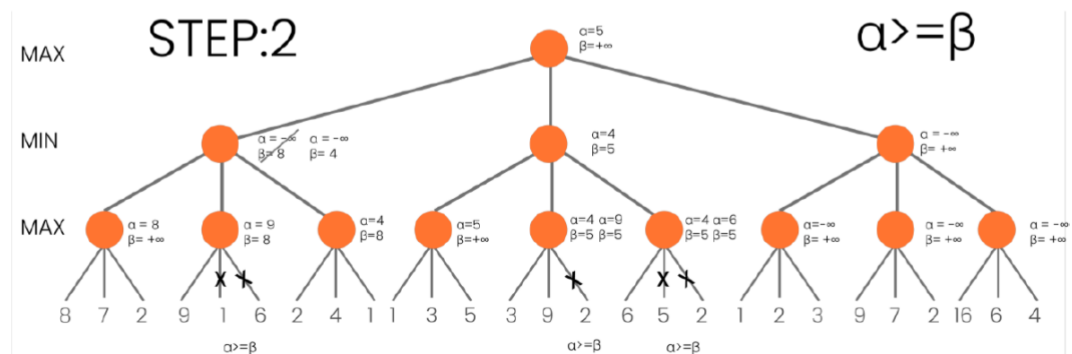
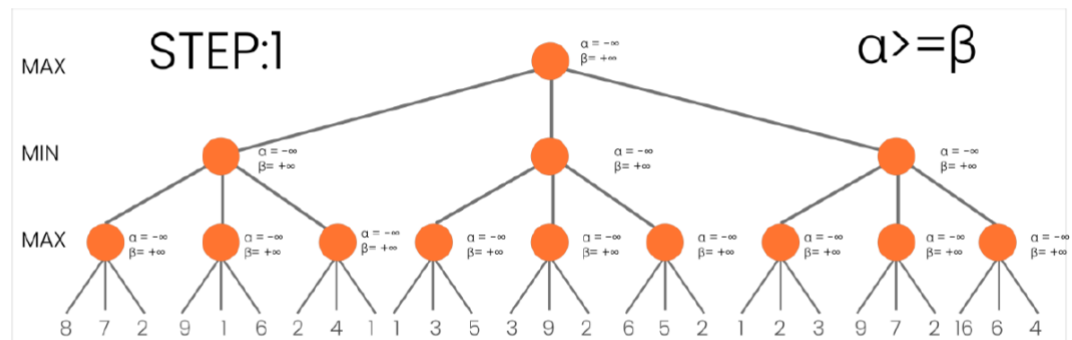
Step 8: C now returns the value of 1 to A here the best value for A is $\max(3, 1) = 3$. Following is the final game tree which is showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



1. Use Min Max with Alpha Beta pruning to decide which path will be taken from the root node. Mark the nodes which will NOT be evaluated. Also show the alpha beta values at all the nodes which are visited. (02 marks)



Answer:



Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

- i. **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- ii. **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Introduction to knowledge

An intelligent agent needs **knowledge** about the real world to take decisions and **reasoning** to act efficiently. Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions**. These agents can represent the world with some formal representation and act intelligently.

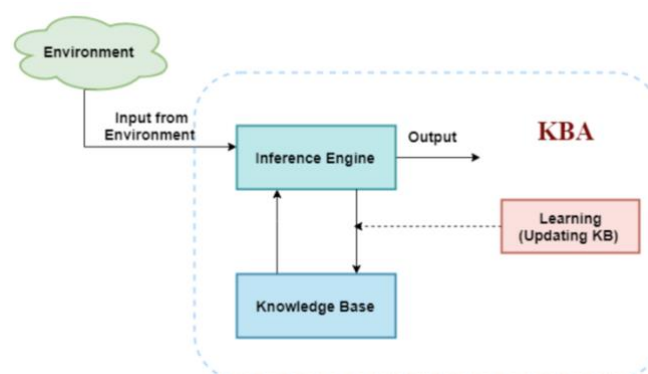
Knowledge-based agents are composed of two main parts:

- A. Knowledge-base and
- B. Inference system.

A knowledge-based agent must be able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent should be able to incorporate new precepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

The architecture of knowledge-based agent:



The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) takes input from the environment by perceiving the environment. The input is taken by the inference/deduction/implication engine of the agent, which also communicates with KB to decide as per the knowledge stored in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

Knowledge base: Knowledge base is a central component of a knowledge-based agent, it is also known as KB. It is a **collection of sentences** (here 'sentence' is a technical term, and it is not identical to a sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores facts about the world.

Why use a knowledge base?

Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

Inference system

Inference means deriving new sentences from old ones. The inference system allows us to add a new sentence to the knowledge base. A sentence is a **proposition/scheme/plan** about the world. The inference system applies logical rules to the KB to deduce new information.

Operations Performed by KBA

The following are three operations which are performed by KBA in order to show intelligent behaviour:

1. **TELL:** This operation tells the knowledge base what it perceives from the environment.
2. **ASK:** This operation asks the knowledge base what action it should perform.
3. **Perform:** It performs the selected action.

A generic knowledge-based agent:

Following is the structure outline of a generic knowledge-based agents' program:

```
function KB-AGENT(percept):  
  persistent: KB, a knowledge base  
    t, a counter, initially 0, indicating time  
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))  
  Action = ASK(KB, MAKE-ACTION-QUERY(t))  
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))  
  t = t + 1  
  return action
```

The knowledge-based agent takes percept as input and returns an action as output. The agent maintains the knowledge base, KB, and it initially has some background knowledge of the real world.

It also has a counter to indicate the time for the whole process, and this counter is initialized with zero.

Each time when the function is called, it performs its three operations:

- A. Firstly, it TELLS the KB what it perceives.
- B. Secondly, it asks KB what action it should take
- C. The third agent program TELLS the KB which action was chosen.

The MAKE-PERCEPT-SENTENCE generates a sentence as a setting that the agent perceived the given percept at the given time.

The MAKE-ACTION-QUERY generates a sentence to ask which action should be done at the current time.

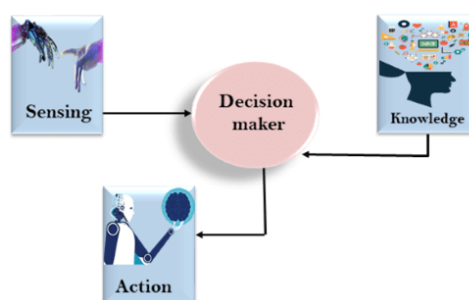
Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and the same in creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behaviour in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose you met some person who is speaking in a language which you don't know, then how you will be able to act on that? The same thing applies to the intelligent behaviour of the agents.

As we can see in the below diagram, there is one decision-maker who acts by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behaviour.

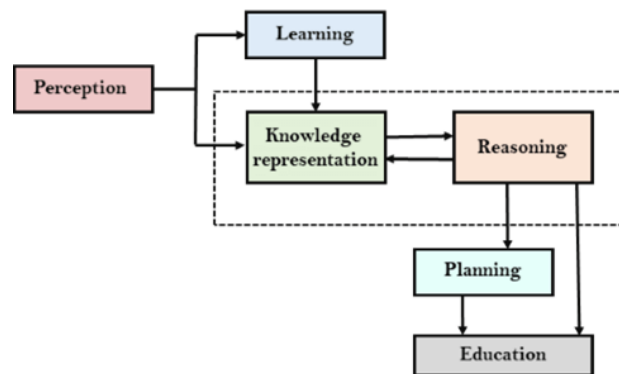


AI knowledge cycle:

An Artificial intelligence system has the following components for displaying intelligent behaviour:

- i. Perception
- ii. Learning
- iii. Knowledge Representation and Reasoning
- iv. Planning

v. Execution



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has a Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception component. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence of machine-like humans. These two components are independent of each other but also coupled together. The planning and execution depend on the analysis of Knowledge representation and reasoning.

Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

- i. **Negation:** A sentence such as $\neg P$ is called the negation of P. A literal can be either Positive literal or negative literal.
- ii. **Conjunction:** A sentence which has \wedge a connective such as, $P \wedge Q$ is called a conjunction.
Example: Rohan is intelligent and hardworking. It can be written as, **P= Rohan is intelligent, Q= Rohan is hardworking. $\rightarrow P \wedge Q$.**
- iii. **Disjunction:** A sentence which has \vee a connective, such as $P \vee Q$. is called disjunction, where P and Q are the propositions. **Example: "Ritika is a doctor or Engineer"**, Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **$P \vee Q$.**
- iv. **Implication:** A sentence such as $P \rightarrow Q$, is called an implication. Implications are also known as if-then rules. It can be represented as **If** it is raining, then the street is wet. Let P= It is raining, and Q= Street is wet, so it is represented as $P \rightarrow Q$
- v. **Biconditional:** A sentence such as $P \Leftrightarrow Q$ is a **Biconditional sentence**, for example **If I am breathing, then I am alive** P= I am breathing, Q= I am alive, it can be represented as $P \Leftrightarrow Q$.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
\wedge	AND	Conjunction	$A \wedge B$
\vee	OR	Disjunction	$A \vee B$
\rightarrow	Implies	Implication	$A \rightarrow B$
\Leftrightarrow	If and only if	Biconditional	$A \Leftrightarrow B$
\neg or \sim	Not	Negation	$\neg A$ or $\neg B$

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
\wedge	AND	Conjunction	$A \wedge B$
\vee	OR	Disjunction	$A \vee B$
\rightarrow	Implies	Implication	$A \rightarrow B$
\Leftrightarrow	If and only if	Biconditional	$A \Leftrightarrow B$
\neg or \sim	Not	Negation	$\neg A$ or $\neg B$

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:

For Negation:

P	$\neg P$
True	False
False	True

For Conjunction:

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

For disjunction:

P	Q	$P \vee Q$
True	True	True
False	True	True
True	False	True
False	False	False

For Implication:

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

For Biconditional:

P	Q	$P \leftrightarrow Q$
True	True	True
True	False	False
False	True	False
False	False	True

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	$\neg R$	$P \vee Q$	$P \vee Q \rightarrow \neg R$
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Logical equivalence:

Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

Let's take two propositions A and B, so for logical equivalence, we can write it as $A \Leftrightarrow B$. In below truth table we can see that column for $\neg A \vee B$ and $A \rightarrow B$, are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators:

i. **Commutativity:**

A. $P \wedge Q = Q \wedge P$, or

B. $P \vee Q = Q \vee P$.

ii. **Associativity:**

A. $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$,

B. $(P \vee Q) \vee R = P \vee (Q \vee R)$

iii. **Identity element:**

A. $P \wedge \text{True} = P$,

B. $P \vee \text{True} = \text{True}$.

iv. **Distributive:**

A. $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$.

B. $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$.

v. **DE Morgan's Law:**

A. $\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$

B. $\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$.

vi. **Double-negation elimination:**

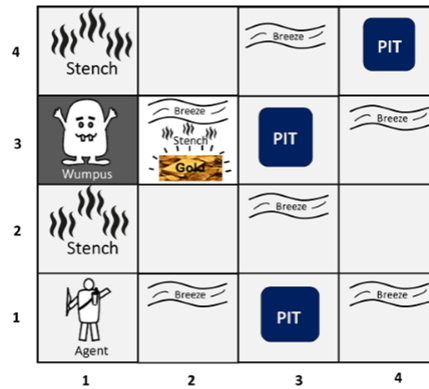
A. $\neg (\neg P) = P$.

The Wumpus World in Artificial intelligence

The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by the video game **Hunt the Wumpus** by Gregory Yob in 1973.

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there is a total of 16 rooms which are connected to each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if an agent falls into Pits, then he will be stuck there forever. The exciting thing about this cave is that in one room there is a possibility of finding a heap of gold. So, the agent's goal is to find the gold and climb out of the cave without falling into Pits or being eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls into the pit (bottom).

Following is a sample diagram for representing the Wumpus world. It is showing some rooms with Pits, one room with Wumpus and one agent at (1, 1) square located in the world.



There are also some components which can help the agent to navigate the cave. These components are given as follows:

- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- 1000 points penalty for being eaten by the Wumpus or falling into the pit.
- 1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environment:

- A 4*4 grid of rooms.
- The agent is initially in room square [1, 1], facing toward the right.
- The location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with a probability 0.2 except the first square.

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

Sensors:

- i. The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- ii. The agent will perceive a **breeze** if he is in the room directly adjacent to the Pit.
- iii. The agent will perceive the **glitter** in the room where the gold is present.
- iv. The agent will perceive the **bump** if he walks into a wall.
- v. When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.
- vi. These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- vii. Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as: **[Stench, Breeze, None, None, None]**.

The Wumpus world Properties:

- i. **Partially observable:** The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- ii. **Deterministic:** It is deterministic, as the result and outcome of the world are already known.
- iii. **Sequential:** The order is important, so it is sequential.
- iv. **Static:** It is static as Wumpus and Pits are not moving.
- v. **Discrete:** The environment is discrete.
- vi. **One agent:** The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

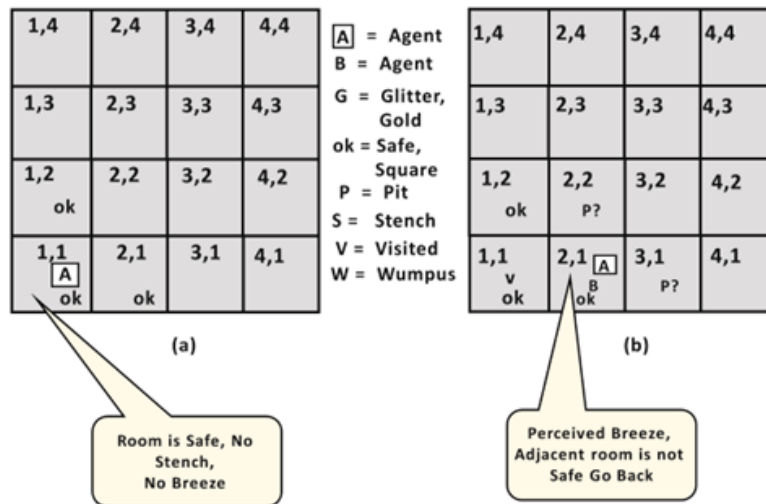
Exploring the Wumpus world:

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add the symbol OK. Symbol A is used to represent the agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, and W for Wumpus.

At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.



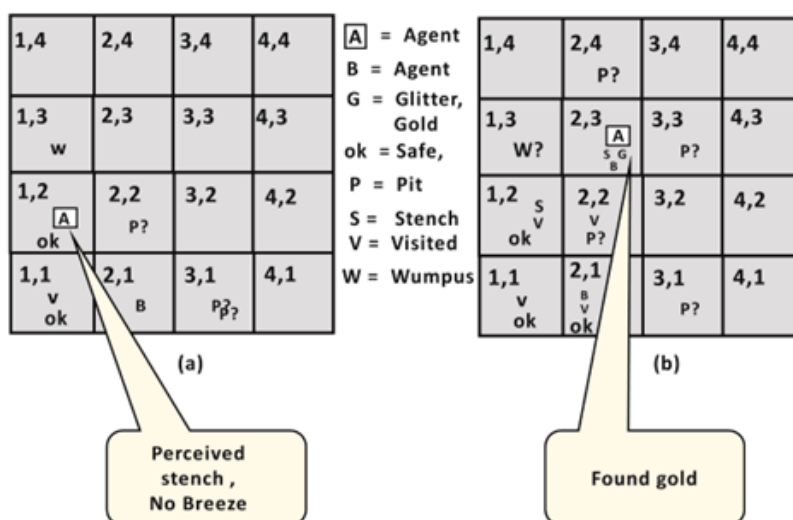
Agent's Second Step:

Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose the agent moves to the room [2, 1], at this room agent perceives some breeze which means the Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add the symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful moves. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use the symbol V to represent the visited squares.

Agent's Third Step:

In the third step, now agent will move to room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by the rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore, the agent infers that Wumpus is in the room [1,3], and in the current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].



Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3].
At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Knowledge base for Wumpus world

As in the previous topic we have learned about the Wumpus world and how a knowledge-based agent evolves the world. Now in this topic, we will create a knowledge base for the Wumpus world and will derive some proof for the Wumpus world using propositional logic.

The agent starts visiting from the first square [1, 1], and we already know that this room is safe for the agent. To build a knowledge base for the Wumpus world, we will use some rules and atomic propositions. We need the symbol [i, j] for each location in the Wumpus world, where i is for the location of rows, and j for column location.

1,4	2,4 P?	3,4	4,4
1,3 W?	2,3 S G B	3,3	4,3
1,2	2,2 V P?	3,2	4,2
1,1 A ok	2,1 B V ok	3,1 P?	4,1

Atomic proposition variable for Wumpus world:

- Let $P_{i,j}$ be true if there is a Pit in the room [i, j].
- Let $B_{i,j}$ be true if the agent perceives breeze in [i, j], (dead or alive).
- Let $W_{i,j}$ be true if there is Wumpus in the square[i, j].
- Let $S_{i,j}$ be true if the agent perceives a stench in the square [i, j].
- Let $V_{i,j}$ be true if that square[i, j] is visited.
- Let $G_{i,j}$ be true if there is gold (and glitter) in the square [i, j].
- Let $OK_{i,j}$ be true if the room is safe.

Note: For a 4 * 4 square board, there will be 7*4*4= 122 propositional variables.

Some Propositional Rules for the Wumpus world:

$$(R1) \neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$$

$$(R2) \neg S_{21} \rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$$

$$(R3) \neg S_{12} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}$$

$$(R4) S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$$

Representation of Knowledgebase for Wumpus world:

Following is the Simple KB for Wumpus world when an agent moves from room [1, 1], to room [2,1]:

$\neg W_{11}$	$\neg S_{11}$	$\neg P_{11}$	$\neg B_{11}$	$\neg G_{11}$	V_{11}	OK_{11}
$\neg W_{12}$	----	$\neg P_{12}$	-----	----	$\neg V_{12}$	OK_{12}
$\neg W_{21}$	$\neg S_{21}$	$\neg P_{21}$	B_{21}	$\neg G_{21}$	V_{21}	OK_{21}

Here in the first row, we have mentioned propositional variables for room [1,1], which is showing that the room does not have Wumpus ($\neg W_{11}$), no stench ($\neg S_{11}$), no Pit ($\neg P_{11}$), no breeze ($\neg B_{11}$), no gold ($\neg G_{11}$), visited (V_{11}), and the room is Safe (OK_{11}).

In the second row, we have mentioned propositional variables for room [1,2], which is showing that there is no Wumpus, stench and breeze are unknown as an agent has not visited room [1,2], no Pit, not visited yet, and the room is safe.

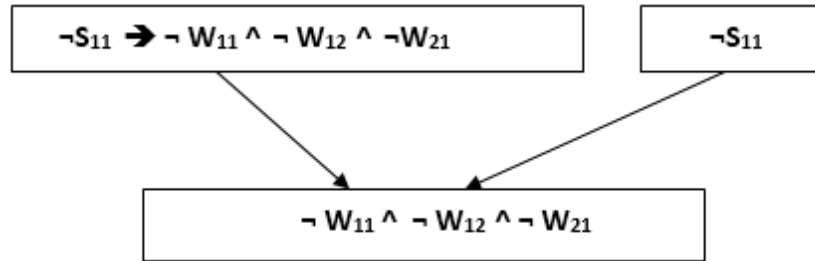
In the third row we have mentioned the propositional variable for room [2,1], which is showing that there is no Wumpus ($\neg W_{21}$), no stench ($\neg S_{21}$), no Pit ($\neg P_{21}$), Perceives breeze (B_{21}), no glitter ($\neg G_{21}$), visited (V_{21}), and the room is safe (OK_{21}).

Prove that Wumpus is in the room (1, 3)

We can prove that Wumpus is in the room (1, 3) using propositional rules which we have derived for the Wumpus world and using inference rule.

- **Apply Modus Ponens** (Latin for "method of affirming." A rule of inference used to draw logical conclusions, which states that if p is true, and if p implies q (p. q), then q is true) **with $\neg S_{11}$ and R1:**

We will firstly apply MP rule with R1 which is $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$, and $\neg S_{11}$ which will give the output $\neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$.

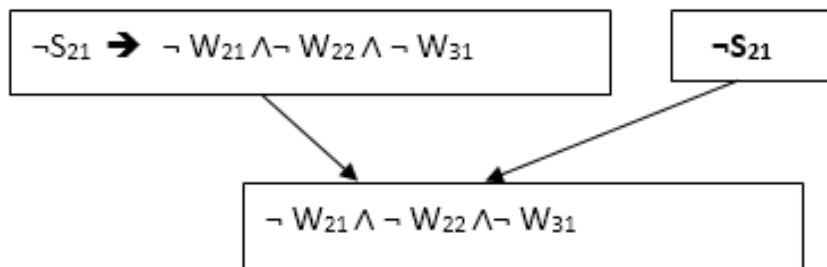


- **Apply And-Elimination Rule:**

After applying And-elimination rule to $\neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$, we will get three statements: $\neg W_{11}$, $\neg W_{12}$, and $\neg W_{21}$.

- **Apply Modus Ponens to $\neg S_{21}$, and R2:**

Now we will apply Modus Ponens to $\neg S_{21}$ and R2 which is $\neg S_{21} \rightarrow \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$, which will give the Output as $\neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

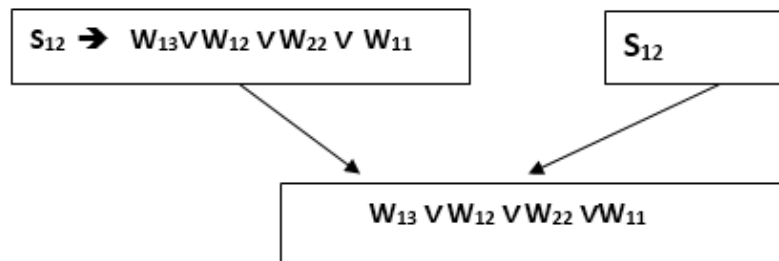


- **Apply And -Elimination rule:**

Now again apply And-elimination rule to $\neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$, We will get three statements: $\neg W_{21}$, $\neg W_{22}$, and $\neg W_{31}$.

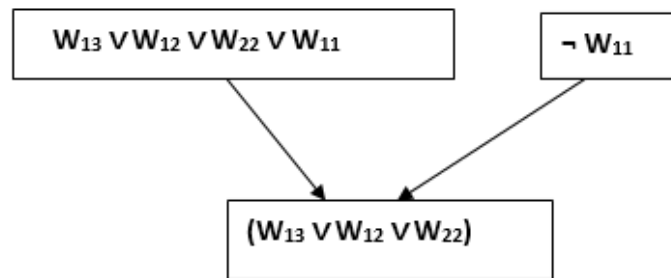
- **Apply MP to S_{12} and R4:**

Apply Modus Ponens to S_{12} and R4 which is $S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$, we will get the output as $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$.



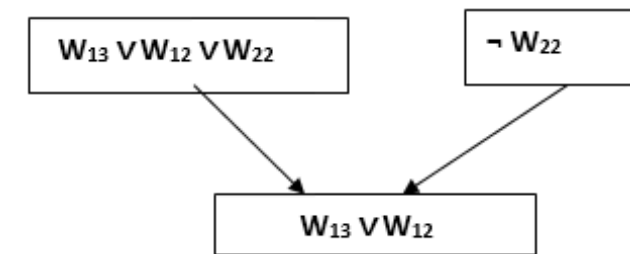
- **Apply Unit resolution on $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ and $\neg W_{11}$:**

After applying Unit resolution formula on $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ and $\neg W_{11}$ we will get $W_{13} \vee W_{12} \vee W_{22}$.



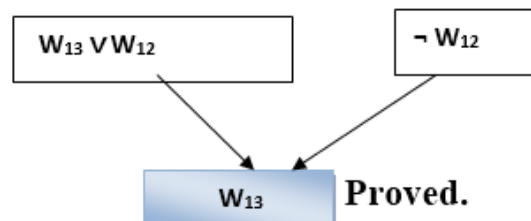
- **Apply Unit resolution on $W_{13} \vee W_{12} \vee W_{22}$ and $\neg W_{22}$:**

After applying Unit resolution on $W_{13} \vee W_{12} \vee W_{22}$, and $\neg W_{22}$, we will get $W_{13} \vee W_{12}$ as output.



- **Apply Unit Resolution on $W_{13} \vee W_{12}$ and $\neg W_{12}$:**

After Applying Unit resolution on $W_{13} \vee W_{12}$ and $\neg W_{12}$, we will get W_{13} as an output, hence it is proved that the Wumpus is in the room [1, 3].



First order Logic

Formalizing English Sentences in FOL

Examples

- There is a student who is loved by every other student.
 $\exists x.(Student(x) \wedge \forall y.(Student(y) \wedge \neg(x = y) \rightarrow Loves(y, x)))$
- Bill is a student.
 $Student(Bill)$
- Bill takes either Analysis or Geometry (but not both).
 $Takes(Bill, Analysis) \leftrightarrow \neg Takes(Bill, Geometry)$
- Bill takes Analysis and Geometry.
 $Takes(Bill, Analysis) \wedge Takes(Bill, Geometry)$
- Bill doesn't take Analysis.
 $\neg Takes(Bill, Analysis)$

6 / 20

Examples

- $bought(Frank, dvd)$
"Frank bought a dvd."
- $\exists x.bought(Frank, x)$
"Frank bought something."
- $\forall x.(bought(Frank, x) \rightarrow bought(Susan, x))$
"Susan bought everything that Frank bought."
- $\forall x.bought(Frank, x) \rightarrow \forall x.bought(Susan, x)$
"If Frank bought everything, so did Susan."
- $\forall x \exists y.bought(x, y)$
"Everyone bought something."
- $\exists x \forall y.bought(x, y)$
"Someone bought everything."

Formalizing English Sentences in FOL

Examples

- All Students are smart.
 $\forall x.(Student(x) \rightarrow Smart(x))$
- There exists a student.
 $\exists x.Student(x)$
- There exists a smart student
 $\exists x.(Student(x) \wedge Smart(x))$
- Every student loves some student
 $\forall x.(Student(x) \rightarrow \exists y.(Student(y) \wedge Loves(x, y)))$
- Every student loves some other student.
 $\forall x.(Student(x) \rightarrow \exists y.(Student(y) \wedge \neg(x = y) \wedge Loves(x, y)))$

5 / 20

Formalizing English Sentences in FOL

Examples

- No students love Bill.
 $\neg \exists x.(Student(x) \wedge Loves(x, Bill))$
- Bill has at least one sister.
 $\exists x.SisterOf(x, Bill)$
- Bill has no sister.
 $\neg \exists x.SisterOf(x, Bill)$
- Bill has at most one sister.
 $\forall x \forall y.(SisterOf(x, Bill) \wedge SisterOf(y, Bill) \rightarrow x = y)$
- Bill has (exactly) one sister.
 $\exists x.(SisterOf(x, Bill) \wedge \forall y.(SisterOf(y, Bill) \rightarrow x = y))$
- Bill has at least two sisters.
 $\exists x \exists y.(SisterOf(x, Bill) \wedge SisterOf(y, Bill) \wedge \neg(x = y))$

7 / 20

Formalizing English Sentences in FOL

Examples

- Every student takes at least one course.
 $\forall x.(Student(x) \rightarrow \exists y.(Course(y) \wedge Takes(x, y)))$
- Only one student failed Geometry.
 $\exists x.(Student(x) \wedge Failed(x, Geometry) \wedge \forall y.(Student(y) \wedge Failed(y, Geometry) \rightarrow x = y))$
- No student failed Geometry but at least one student failed Analysis.
 $\neg \exists x.(Student(x) \wedge Failed(x, Geometry)) \wedge \exists x.(Student(x) \wedge Failed(x, Analysis))$
- Every student who takes Analysis also takes Geometry.
 $\forall x.(Student(x) \wedge Takes(x, Analysis) \rightarrow Takes(x, Geometry))$

8 / 20

Bayesian Belief Network in artificial intelligence

Bayesian belief network is key computer technology for dealing with probabilistic events and solving a problem which has uncertainty. We can define a Bayesian network as: "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph." It is also called a **Bayes network**, **belief network**, **decision network**, or **Bayesian model**. Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

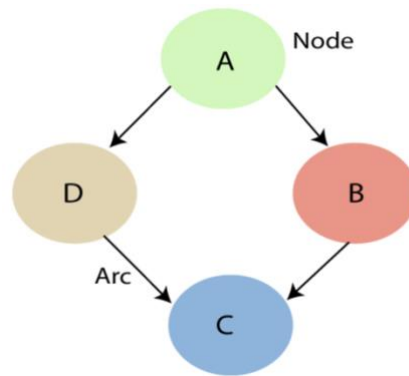
Real-world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction**, **anomaly detection**, **diagnostics**, **automated insight**, **reasoning**, **time series prediction**, and decision-making under uncertainty.

Bayesian Network can be used for building models from data and experts' opinions, and it consists of two parts:

- Directed Acyclic Graph**
- Table of conditional probabilities.**

The generalized form of a Bayesian network that represents and solves decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**. **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influences the other node, and if there is no directed link that means that nodes are independent of each other

- i. In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.
- ii. If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.
- iii. Node C is independent of node A.

The Bayesian network has mainly two components:

- i. **Causal Component**
- ii. **Actual numbers**

Each node in the Bayesian network has condition probability distribution $P(X_i | \text{Parent}(X_i))$, which determines the effect of the parent on that node. Bayesian network is based on Joint probability distribution and conditional probability. So, let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables $x_1, x_2, x_3, \dots, x_n$, then the probabilities of a different combination of $x_1, x_2, x_3, \dots, x_n$, are known as Joint probability distribution.

$P[x_1, x_2, x_3, \dots, x_n]$, it can be written as the following way in terms of the joint probability distribution.

$$= P[x_1 | x_2, x_3, \dots, x_n] P[x_2, x_3, \dots, x_n]$$

$$= P[x_1 | x_2, x_3, \dots, x_n] P[x_2 | x_3, \dots, x_n] \dots P[x_{n-1} | x_n] P[x_n].$$

In general, for each variable X_i , we can write the equation as:

$$P(X_i | X_{i-1}, \dots, X_1) = P(X_i | \text{Parents}(X_i))$$

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds to detecting a burglary but also responds to minor earthquakes. Harry has two neighbours David and Sophia, who have taken responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses hearing the alarm. Here we would like to compute the probability of a Burglary Alarm.

Problem: Calculate the probability that the alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry

Solution:

- i. The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affect the probability of the alarm going off, but David and Sophia's calls depend on alarm probability.
- ii. The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also do not confer before calling.
- iii. The conditional distributions for each node are given as conditional probabilities tables or CPT.
- iv. Each row in the CPT must be summed to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- v. In CPT, a Boolean variable with k Boolean parents contains 2^k probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- i. **Burglary (B)**
- ii. **Earthquake(E)**
- iii. **Alarm(A)**
- iv. **David Calls(D)**
- v. **Sophia calls(S)**

We can write the events of the problem statement in the form of probability: $P[D, S, A, B, E]$, can rewrite the above probability statement using joint probability distribution:

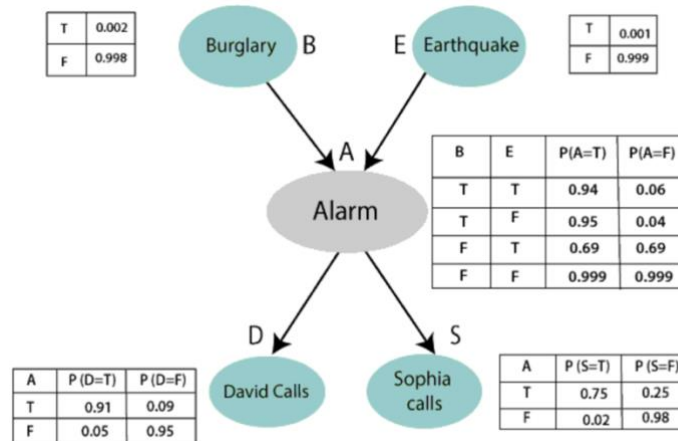
$$P[D, S, A, B, E] = P[D | S, A, B, E] \cdot P[S, A, B, E]$$

$$= P[D | S, A, B, E] \cdot P[S | A, B, E] \cdot P[A, B, E]$$

$$= P[D | A] \cdot P[S | A, B, E] \cdot P[A, B, E]$$

$$= P[D | A]. P[S | A]. P[A | B, E]. P[B, E]$$

$$= P[D | A]. P[S | A]. P[A | B, E]. P[B | E]. P[E]$$



Let's take the observed probability for the Burglary and earthquake component:

$P(B = \text{True}) = 0.002$, which is the probability of burglary.

$P(B = \text{False}) = 0.998$, which is the probability of no burglary.

$P(E = \text{True}) = 0.001$, which is the probability of a minor earthquake

$P(E = \text{False}) = 0.999$, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A = True)	P(A = False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$$P(S, D, A, \neg B, \neg E) = P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E).$$

$$= 0.75 * 0.91 * 0.001 * 0.998 * 0.999$$

$$= 0.00068045.$$

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

The semantics of the Bayesian Network:

There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution.

It is helpful to understand how to construct the network.

2. To understand the network as an encoding of a collection of conditional independence statements.

It is helpful in designing inference procedures.

Numerical Example: On the given dataset, train a Naïve Bays algorithm to find the probability whether Tennis will be played on a particular day with given characteristics "Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong"

NAIVE BAYES CLASSIFIER – Example -1

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

(Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong)

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

$$P(\text{PlayTennis} = \text{yes}) = 9/14 = .64$$

$$P(\text{PlayTennis} = \text{no}) = 5/14 = .36$$

NAIVE BAYES CLASSIFIER Example - 1

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

$$P(\text{PlayTennis} = \text{yes}) = 9/14 = .64$$

$$P(\text{PlayTennis} = \text{no}) = 5/14 = .36$$

Outlook	Y	N	Humidity	Y	N
sunny	2/9	3/5	high	3/9	4/5
overcast	4/9	0	normal	6/9	1/5
rain	3/9	2/5			
Temperature			Wind		
hot	2/9	2/5	Strong	3/9	3/5
mild	4/9	2/5	Weak	6/9	2/5
cool	3/9	1/5			

NAIVE BAYES CLASSIFIER Example - 1

NAIVE BAYES CLASSIFIER – Example -1

$\langle \text{Outlook} = \text{sunny}, \text{Temperature} = \text{cool}, \text{Humidity} = \text{high}, \text{Wind} = \text{strong} \rangle$

$$v_{NB} = \underset{v_j \in \{\text{yes}, \text{no}\}}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

$$= \underset{v_j \in \{\text{yes}, \text{no}\}}{\operatorname{argmax}} P(v_j) P(\text{Outlook} = \text{sunny} | v_j) P(\text{Temperature} = \text{cool} | v_j) \\ \cdot P(\text{Humidity} = \text{high} | v_j) P(\text{Wind} = \text{strong} | v_j)$$

$$v_{NB}(\text{yes}) = P(\text{yes}) P(\text{sunny} | \text{yes}) P(\text{cool} | \text{yes}) P(\text{high} | \text{yes}) P(\text{strong} | \text{yes}) = .0053$$

$$v_{NB}(\text{no}) = P(\text{no}) P(\text{sunny} | \text{no}) P(\text{cool} | \text{no}) P(\text{high} | \text{no}) P(\text{strong} | \text{no}) = .0206$$

$$v_{NB}(\text{yes}) = \frac{v_{NB}(\text{yes})}{v_{NB}(\text{yes}) + v_{NB}(\text{no})} = 0.205 \quad v_{NB}(\text{no}) = \frac{v_{NB}(\text{no})}{v_{NB}(\text{yes}) + v_{NB}(\text{no})} = 0.795$$

Question 2: Predict Bayes algorithm weather we can pet an animal or not. Find the probability of petting animal test = (Dog Medium White).

	Animals	Size of Animal	Body Color	Can we Pet them
0	Dog	Medium	Black	Yes
1	Dog	Big	White	No
2	Rat	Small	White	Yes
3	Cow	Big	White	Yes
4	Cow	Small	Brown	No
5	Cow	Big	Black	Yes
6	Rat	Big	Brown	No
7	Dog	Small	Brown	Yes
8	Dog	Medium	Brown	Yes
9	Cow	Medium	White	No
10	Dog	Small	Black	Yes
11	Rat	Medium	Black	No
12	Rat	Small	Brown	No
13	Cow	Big	White	Yes

Solution:

Animals				
	Yes	No	P(Yes)	P(No)
Dog	4	1	4/8	1/6
Rat	1	3	1/8	3/6
Cow	3	2	3/8	2/6
Total	8	6	100%	100%

Size of Animal				
	Yes	No	P(Yes)	P(No)
Medium	2	2	2/8	2/6
Big	3	2	3/8	2/6
Small	3	2	3/8	2/6
Total	8	6	100%	100%

Body Color				
	Yes	No	P(Yes)	P(No)
Black	3	1	3/8	1/6
White	3	2	3/8	2/6
Brown	2	3	2/8	3/6
Total	8	6	100%	100%

$$P(\text{yes}) = 8/14$$

$$P(\text{no}) = 6/14$$

Petting animal test = (Dog, Medium, White)

$$P(\text{yes} | \text{Dog Medium White}) = P(\text{yes}) * p(\text{dog} | \text{yes}) * p(\text{white} | \text{yes}) * p(\text{Medium} | \text{yes})$$

$$= 8/14 * 4/8 * 3/8 * 2/8$$

$$= 0.026$$

$$P(\text{no} | \text{Dog Medium White}) = P(\text{yes}) * p(\text{dog} | \text{no}) * p(\text{white} | \text{no}) * p(\text{Medium} | \text{no})$$

$$= 6/14 * 1/6 * 2/6 * 2/6$$

$$= 0.0075$$

$$P(\text{yes}) = 0.026 / (0.026 + 0.0075)$$

$$= 0.78$$

$$= 78\%$$

$$P(\text{no}) = 0.0075 / (0.026 + 0.0075)$$

$$= 0.22$$

$$= 22\%$$

Result:

We can take animal as a pet.

Question 2: Predict Bayes algorithm weather we can pet an animal or not. Find the probability of petting animal test = (Cow Medium Black).

	Animals	Size of Animal	Body Color	Can we Pet them
0	Dog	Medium	Black	Yes
1	Dog	Big	White	No
2	Rat	Small	White	Yes
3	Cow	Big	White	Yes
4	Cow	Small	Brown	No
5	Cow	Big	Black	Yes
6	Rat	Big	Brown	No
7	Dog	Small	Brown	Yes
8	Dog	Medium	Brown	Yes
9	Cow	Medium	White	No
10	Dog	Small	Black	Yes
11	Rat	Medium	Black	No
12	Rat	Small	Brown	No
13	Cow	Big	White	Yes

Solution:

Animals				
	Yes	No	P(Yes)	P(No)
Dog	4	1	4/8	1/6
Rat	1	3	1/8	3/6
Cow	3	2	3/8	2/6
Total	8	6	100%	100%

Size of Animal				
	Yes	No	P(Yes)	P(No)
Medium	2	2	2/8	2/6
Big	3	2	3/8	2/6
Small	3	2	3/8	2/6
Total	8	6	100%	100%

Body Color				
	Yes	No	P(Yes)	P(No)
Black	3	1	3/8	1/6
White	3	2	3/8	2/6
Brown	2	3	2/8	3/6
Total	8	6	100%	100%

$$P(\text{yes}) = 8/14$$

$$P(\text{no}) = 6/14$$

$$\text{Petting animal test} = (\text{Cow}, \text{Medium}, \text{Black})$$

$$P(\text{yes} | \text{Cow Medium Black}) = P(\text{yes}) * p(\text{Cow} | \text{yes}) * p(\text{Medium} | \text{yes}) * p(\text{Black} | \text{yes})$$

$$= 8/14 * 3/8 * 2/8 * 2/8$$

$$= 0.013$$

$$P(\text{no} | \text{Dog Medium White}) = P(\text{yes}) * p(\text{dog} | \text{no}) * p(\text{white} | \text{no}) * p(\text{Medium} | \text{no})$$

$$= 6/14 * 2/6 * 2/6 * 1/6$$

$$= 0.0079$$

Normalization:

$$P(\text{yes}) = 0.013 / (0.013 + 0.0079)$$

$$= 0.62$$

$$= 62\%$$

$$P(\text{no}) = 0.0079 / (0.013 + 0.0079)$$

$$= 0.38$$

$$= 38\%$$

Result:

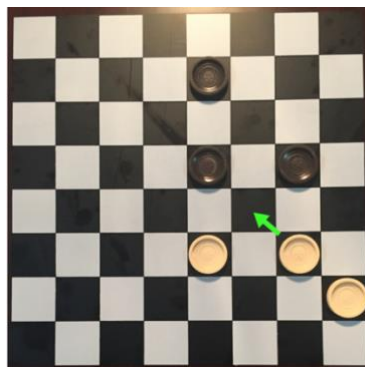
We can take animal as a pet

Question: If we are to create an AI-based problem solver for “game of checkers”, answer the following:

- a) How would you represent the problem and why
- b) What search strategy will you use and why?
- c) What heuristic value will you use and why?
- d) How will you use predicates and/or propositions to describe any state?
- e) Does this involve uncertainty and planning, and how?

Answer:

Checkers, also known as draughts, is a board game played on an 8x8 board with 12 pieces per player. The game is played between two players, each starting with 12 pieces, placed on the dark squares of the board. The objective of the game is to capture all of the opponent's pieces or to block the opponent's pieces so they can't move. Each player's pieces can only move diagonally, and pieces that reach the opponent's end of the board are "kinged" and can move both forwards and backward. The game is won by either capturing all of the opponent's pieces or trapping their pieces so they cannot move.



a) To create an AI-based problem solver for the game of checkers, one possible representation of the problem is a state space search problem. This would involve modelling the game as a state space, with the current state of the board as the root node and legal moves as the edges connecting the nodes. The goal state would be represented as a win or loss for one of the players. The state can be represented by a tuple which contains the following information:

- The current board configuration is represented as an 8x8 array of integers, with each integer representing a type of piece or an empty square.
 - The current player, represented as an integer (1 for white, -1 for black)
 - The set of legal moves for the current player represented as a list of tuples, where each tuple represents the starting and ending coordinates of a move
- b) The search strategy we will use is the Minimax algorithm with Alpha-beta pruning, as it has proven to be effective in solving two-player games such as checkers. The Minimax algorithm simulates the decision-making process of the opponent, and it "looks ahead" a certain number of moves. Alpha-beta pruning is used to improve the efficiency of the search process by reducing the number of nodes that need to be explored, and thus, making the algorithm much more efficient.
 - c) The heuristic value we will use is a combination of Material advantage, Mobility, King-row control, Centre control, and Pawn structure. These heuristics can evaluate different aspects of the game such as the number of pieces on the board, legal moves available, pieces close to becoming kinged, and control of the central squares and pawn structure, respectively. The correct weights of these heuristics are usually determined by testing the algorithm against a large number of training games.
 - d) To describe any state, we will use predicates and propositions as a formal language to express the current state and the possible actions, making it easier to reason about them and make

decisions based on that information. Examples of predicates that can be used to describe the state of the game include: "Square (x,y) is occupied by a white piece", "Piece at (x1,y1) can capture a piece at (x2,y2)", and "Player X has a capture move".

- e) e) Yes, uncertainty and planning are involved in the game of checkers. The opponent's moves are not known in advance, which creates uncertainty. Planning is required as the AI player must consider multiple possible moves and evaluate their potential outcomes. The AI uses search algorithms such as the Minimax algorithm with Alpha-beta pruning to simulate the decision-making process of the opponent and "look ahead" a certain number of moves, and also uses the evaluation function to score different board positions and plan for the most likely outcomes.

Question: Comment, with example(s), whether home is an example of Multi Agent System or not.

Whether a home is an example of a Multi-Agent System (MAS) depends on the level of autonomy and communication of the devices and systems within it. A home is a physical environment where various devices and systems, such as heating and cooling systems, lighting, security systems, and appliances, are used to provide comfort and convenience for the inhabitants. Many of these devices and systems can be controlled by a central system, such as a smart home system, which allows for automation and remote control of the devices and systems. However, not all smart homes can be considered as Multi-Agent Systems. If the devices and systems in a home are only able to communicate with a central system and are not autonomous, they are not considered agents, since they are not autonomous entities that can make their own decisions and take actions on their own, but instead, they are dependent on the commands of the central system. On the other hand, when these systems are autonomous, able to take decisions based on certain parameters, and able to communicate among themselves to coordinate and optimize their actions, it can be considered as a Multi-Agent System. For example, a home that has smart devices like thermostats, lighting, security cameras, and smart appliances which are all connected to the internet and communicate with each other to optimize energy consumption and improve the overall comfort and security of the home. All these devices are autonomous, able to make decisions based on sensor input and adjust their settings, they can communicate with each other to coordinate actions. This home can be considered as a Multi-Agent System. In summary, whether a home is an example of a Multi-Agent System depends on the level of autonomy and communication of the devices and systems within it. If the devices and systems in a home are autonomous, able to communicate with each other and make decisions based on certain parameters, then it can be considered as a Multi-Agent System.

Question: What is CryptArithmetic Problem in Artificial Intelligence

In artificial intelligence, a cryptarithmic problem is a type of mathematical puzzle that involves creating an arithmetic equation using words or letters. Each letter or word represents a different digit, and the goal is to determine the digit that each letter or word represents in order to make the equation a valid mathematical equation. Cryptarithmic problems are often used as a test problem for constraint satisfaction problem and logic programming for example : $SEND + MORE = MONEY$ Here, each letter represents a digit, and the goal is to find a unique mapping of digits to letters such that the equation $SEND + MORE = MONEY$ is true.

Constraint Satisfaction Problem (CSP) is a method of solving problems by specifying the constraints on the variables and finding a solution that satisfies all the constraints. Cryptarithmic is a type of mathematical puzzle in which a set of words or letters is given and the goal is to find the numerical values of the letters such that the mathematical equation is true.

Constraints:

- Each letter represents a unique digit from 0-9
- The leading digit of each number cannot be 0

- NO two letters can have same digit.
- Only 1 carry is forwarded

1. EVER + SINCE = DARWIN

E V E R
 S I N C E

 D A R W I N

E	5
V	6
R	3
S	9
I	7
N	8
C	2
D	1
A	0
W	4

2. HOW + MUCH = POWER

H O W
 M U C H

 P O W E R

H	7
O	0
W	5
M	9
U	8
C	3
H	7
P	1
E	4
R	2

3. DAYS + TOO = SHORT

D	9
A	7
Y	4
S	3
T	5
O	2
H	0

R	6
---	---

DAYS

TOO

SHORT

4. TWO + DAYS = MORE

TWO

DAYS

MORE

T	8
W	9
O	3
D	6
A	4
Y	1
S	2
M	7
R	0
E	5

Automated Planning:

Planning a course of action is a key requirement for an intelligent agent. The right representation for actions and states and the right algorithms can make this easier.

Classical Planning

Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment. We have seen two approaches to this task: the problem-solving agent and the hybrid propositional logical agent. Both share two limitations. First, they both require ad hoc heuristics for each new domain: a heuristic evaluation function for search and hand-written code for the hybrid Wumpus agent. Second, they both need to explicitly represent an exponentially large state space. For example, in the propositional logic model of the Wumpus world, the axiom for moving a step forward had to be repeated for all four agent orientations, T time steps, and n^2 current locations.

In response to these limitations, planning researchers have invested in a factored representation using a family of languages called PDDL, the Planning Domain Definition of Language, which allows us to express all n2 actions with a single action schema and does not need domain-specific knowledge. Basic PDDL can handle classical planning domains, and extensions can handle non-classical domains that are continuous, partially observable, concurrent, and multi-agent. The syntax of PDDL is based on Lisp, but we will translate it into a form that matches the notation used in this book.

Example domain: Air cargo transport

Figure 11.1 shows an air cargo transport problem involving loading and unloading cargo and flying it from place to place. The problem can be defined with three actions: Load, Unload, and Fly. The actions affect two predicates: $In(c; p)$ means that cargo c is inside plane p , and $At(x;a)$ means that object x (either plane or cargo) is at airport a . Note that some care must be taken to make sure the At predicates are maintained properly.

```

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK)
    ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2)
    ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
Action(Load(c, p, a),
    PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
    EFFECT: ¬At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
    PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
    EFFECT: At(c, a) ∧ ¬In(c, p))
Action(Fly(p, from, to),
    PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
    EFFECT: ¬At(p, from) ∧ At(p, to))

```

Figure 11.1 A PDDL description of an air cargo transportation planning problem.

When a plane flies from one airport to another, all the cargo inside the plane goes with it. In first-order logic it would be easy to quantify over all objects that are inside the plane. But PDDL does not have a universal quantifier, so we need a different solution. The approach we use is to say that a piece of cargo ceases to be at anywhere when it is *In* a plane; the cargo only becomes *At* the new airport when it is unloaded. So, *At* really means “available for use at a given location.” The following plan is a solution to the problem:

```

[Load (C1, P1, SFO), Fly (P1, SFO, JFK), Unload (C1, P1, JFK),
Load (C2, P2, JFK), Fly (P2, JFK, SFO), Unload (C2, P2, SFO)],

```

Example domain: The spare tire problem

Consider the problem of changing a flat tire (Figure 11.2). The goal is to have a good spare tire properly mounted onto the car’s axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk. To keep it simple, our version of the problem is an abstract one, with no sticky lug nuts or other complications. There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight. We assume that the car is parked in a particularly bad neighbourhood, so that the effect of leaving it overnight is that the tires disappear.

[Remove (Flat, Axle), Remove (Spare, Trunk), PutOn (Spare, Axle)] is a solution to the problem.

```

Init(Tire(Flat)  $\wedge$  Tire(Spare)  $\wedge$  At(Flat,Axle)  $\wedge$  At(Spare,Trunk))
Goal(At(Spare,Axle))
Action(Remove(obj,loc),
  PRECOND: At(obj,loc)
  EFFECT:  $\neg$  At(obj,loc)  $\wedge$  At(obj,Ground))
Action(PutOn(t, Axle),
  PRECOND: Tire(t)  $\wedge$  At(t,Ground)  $\wedge$   $\neg$  At(Flat,Axle)  $\wedge$   $\neg$  At(Spare,Axle)
  EFFECT:  $\neg$  At(t,Ground)  $\wedge$  At(t,Axle))
Action(LeaveOvernight,
  PRECOND:
  EFFECT:  $\neg$  At(Spare,Ground)  $\wedge$   $\neg$  At(Spare,Axle)  $\wedge$   $\neg$  At(Spare,Trunk)
 $\wedge$   $\neg$  At(Flat,Ground)  $\wedge$   $\neg$  At(Flat,Axle)  $\wedge$   $\neg$  At(Flat, Trunk))

```

Figure 11.2 The simple spare tire problem.

Example domain: The blocks world

One of the most famous planning domains is the blocks world. This domain consists of a set of cube-shaped blocks sitting on an arbitrarily-large table.¹ The blocks can be stacked, but only one block can fit directly on top of another. A robot arm can pick up a block and move it to another position, either on the table or on top of another block. The arm can pick up only one block at a time, so it cannot pick up a block that has another one on top of it. A typical goal to get block A on B and block B on C (see Figure 11.3).

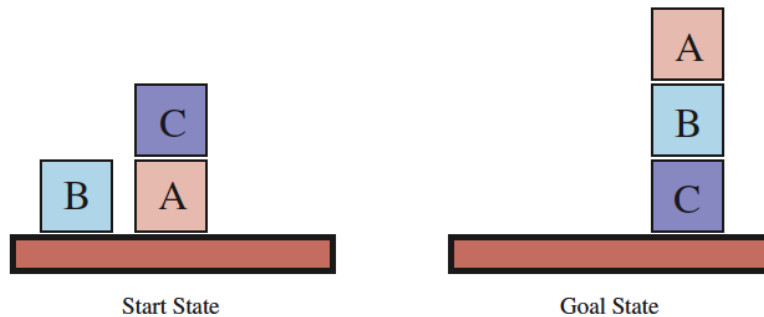


Figure 11.3 Diagram of the blocks-world problem in Figure 11.4.

```

Init(On(A,Table)  $\wedge$  On(B,Table)  $\wedge$  On(C,A)
     $\wedge$  Block(A)  $\wedge$  Block(B)  $\wedge$  Block(C)  $\wedge$  Clear(B)  $\wedge$  Clear(C)  $\wedge$  Clear(Table))
Goal(On(A,B)  $\wedge$  On(B,C))
Action(Move(b,x,y),
    PRECOND: On(b,x)  $\wedge$  Clear(b)  $\wedge$  Clear(y)  $\wedge$  Block(b)  $\wedge$  Block(y)  $\wedge$ 
        (b $\neq$ x)  $\wedge$  (b $\neq$ y)  $\wedge$  (x $\neq$ y),
    EFFECT: On(b,y)  $\wedge$  Clear(x)  $\wedge$   $\neg$ On(b,x)  $\wedge$   $\neg$ Clear(y))
Action(MoveToTable(b,x),
    PRECOND: On(b,x)  $\wedge$  Clear(b)  $\wedge$  Block(b)  $\wedge$  Block(x),
    EFFECT: On(b,Table)  $\wedge$  Clear(x)  $\wedge$   $\neg$ On(b,x))

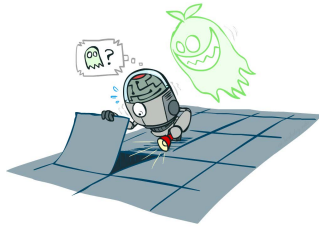
```

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence [*MoveToTable*(*C*,*A*), *Move*(*B*,*Table*,*C*), *Move*(*A*,*Table*,*B*)].

We use *On*(*b*,*x*) to indicate that block *b* is on *x*, where *x* is either another block or the table. The action for moving block *b* from the top of *x* to the top of *y* will be *Move*(*b*,*x*,*y*). Now, one of the preconditions on moving *b* is that no other block be on it. In first-order logic, this would be $\neg\exists x \text{ } On(x,b)$ or, alternatively, $\forall x \neg On(x,b)$. Basic PDDL does not allow quantifiers, so instead we introduce a predicate *Clear*(*x*) that is true when nothing is on *x*. (The complete problem description is in Figure 11.4.)

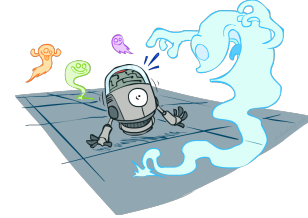
CS188 Outline

- We're done with Part I: Search and Planning!
- Part II: Probabilistic Reasoning
 - Diagnosis
 - Speech recognition
 - Tracking objects
 - Robot mapping
 - Genetics
 - Error correcting codes
 - ... lots more!
- Part III: Machine Learning



CS 188: Artificial Intelligence

Probability

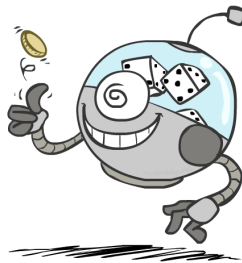


Instructors: Dan Klein and Pieter Abbeel --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.]

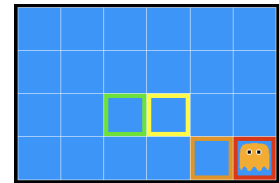
Today

- Probability
 - Random Variables
 - Joint and Marginal Distributions
 - Conditional Distribution
 - Product Rule, Chain Rule, Bayes' Rule
 - Inference
 - Independence
- You'll need all this stuff A LOT for the next few weeks, so make sure you go over it now!



Inference in Ghostbusters

- A ghost is in the grid somewhere
- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



- Sensors are noisy, but we know $P(\text{Color} \mid \text{Distance})$

$P(\text{red} \mid 3)$	$P(\text{orange} \mid 3)$	$P(\text{yellow} \mid 3)$	$P(\text{green} \mid 3)$
0.05	0.15	0.5	0.3

[Demo: Ghostbuster – no probability (L12D1)]

Uncertainty

- General situation:
 - **Observed variables (evidence):** Agent knows certain things about the state of the world (e.g., sensor readings or symptoms)
 - **Unobserved variables:** Agent needs to reason about other aspects (e.g. where an object is or what disease is present)
 - **Model:** Agent knows something about how the known variables relate to the unknown variables
- Probabilistic reasoning gives us a framework for managing our beliefs and knowledge

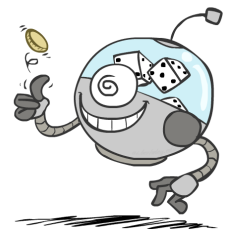
0.11	0.11	0.11
0.11	0.11	0.11
0.11	0.11	0.11

0.17	0.10	0.10
0.09	0.13	0.10
0.01	0.09	0.17

-0.01	-0.01	-0.01
-0.01	0.05	0.05
0.01	0.05	-0.01

Random Variables

- A random variable is some aspect of the world about which we (may) have uncertainty
 - R = Is it raining?
 - T = Is it hot or cold?
 - D = How long will it take to drive to work?
 - L = Where is the ghost?
- We denote random variables with capital letters
- Like variables in a CSP, random variables have domains
 - R in {true, false} (often write as {+, -})
 - T in {hot, cold}
 - D in $[0, \infty)$
 - L in possible locations, maybe $\{(0,0), (0,1), \dots\}$



Probability Distributions

- Associate a probability with each value

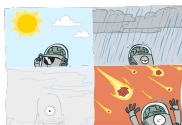
- Temperature:



$$P(T)$$

T	P
hot	0.5
cold	0.5

- Weather:



$$P(W)$$

W	P
sun	0.6
rain	0.1
fog	0.3
meteor	0.0

Probability Distributions

- Unobserved random variables have distributions

$$P(T)$$

T	P
hot	0.5
cold	0.5

$$P(W)$$

W	P
sun	0.6
rain	0.1
fog	0.3
meteor	0.0

Shorthand notation:

$$P(\text{hot}) = P(T = \text{hot}),$$

$$P(\text{cold}) = P(T = \text{cold}),$$

$$P(\text{rain}) = P(W = \text{rain}),$$

$$\dots$$

OK if all domain entries are unique

- A distribution is a TABLE of probabilities of values

- A probability (lower case value) is a single number

$$P(W = \text{rain}) = 0.1$$

- Must have: $\forall x \ P(X = x) \geq 0$ and $\sum_x P(X = x) = 1$

Joint Distributions

- A *joint distribution* over a set of random variables: X_1, X_2, \dots, X_n specifies a real number for each assignment (or *outcome*):

$$P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$$

$$P(x_1, x_2, \dots, x_n)$$

- Must obey: $P(x_1, x_2, \dots, x_n) \geq 0$

$$\sum_{(x_1, x_2, \dots, x_n)} P(x_1, x_2, \dots, x_n) = 1$$

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

- Size of distribution if n variables with domain sizes d?

- For all but the smallest distributions, impractical to write out!

Probabilistic Models

- A probabilistic model is a joint distribution over a set of random variables

- Probabilistic models:

- (Random) variables with domains
- Assignments are called *outcomes*
- Joint distributions: say whether assignments (outcomes) are likely
- Normalized: sum to 1.0
- Ideally: only certain variables directly interact

Distribution over T,W

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

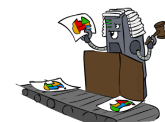


- Constraint satisfaction problems:

- Variables with domains
- Constraints: state whether assignments are possible
- Ideally: only certain variables directly interact

Constraint over T,W

T	W	P
hot	sun	T
hot	rain	F
cold	sun	F
cold	rain	T



Events

- An *event* is a set E of outcomes

$$P(E) = \sum_{(x_1, \dots, x_n) \in E} P(x_1, \dots, x_n)$$

- From a joint distribution, we can calculate the probability of any event

- Probability that it's hot AND sunny?
- Probability that it's hot?
- Probability that it's hot OR sunny?

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

- Typically, the events we care about are *partial assignments*, like $P(T=\text{hot})$

Quiz: Events

- $P(+x, +y)$?

- $P(+x)$?

- $P(-y \text{ OR } +x)$?

$P(X, Y)$

X	Y	P
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

Marginal Distributions

- Marginal distributions are sub-tables which eliminate variables
- Marginalization (summing out): Combine collapsed rows by adding

$P(T, W)$			$P(T)$	
T	W	P	T	P
hot	sun	0.4	hot	0.5
hot	rain	0.1	cold	0.5
cold	sun	0.2		
cold	rain	0.3		

$P(s)$			$P(W)$	
t	s	P	W	P
	sun	0.6	sun	0.6
	rain	0.4	rain	0.4

$$P(X_1 = x_1) = \sum_{x_2} P(X_1 = x_1, X_2 = x_2)$$

Quiz: Marginal Distributions

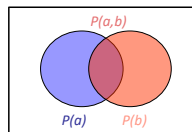
$P(X, Y)$			$P(X)$	
X	Y	P	X	P
+x	+y	0.2	+x	
+x	-y	0.3	-x	
-x	+y	0.4		
-x	-y	0.1		

$P(Y)$			$P(X)$	
Y	P		X	P
+y			+x	
-y			-x	

Conditional Probabilities

- A simple relation between joint and conditional probabilities
- In fact, this is taken as the *definition* of a conditional probability

$$P(a|b) = \frac{P(a, b)}{P(b)}$$



T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$P(W = s|T = c) = \frac{P(W = s, T = c)}{P(T = c)} = \frac{0.2}{0.5} = 0.4$$

$$= P(W = s, T = c) + P(W = r, T = c)$$

$$= 0.2 + 0.3 = 0.5$$

Quiz: Conditional Probabilities

$P(X, Y)$		
X	Y	P
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

- $P(+x | +y)$?
- $P(-x | +y)$?
- $P(-y | +x)$?

Conditional Distributions

- Conditional distributions are probability distributions over some variables given fixed values of others

Conditional Distributions			Joint Distribution		
$P(W T = \text{hot})$			$P(T, W)$		
W	P		T	W	P
sun	0.8		hot	sun	0.4
rain	0.2		hot	rain	0.1
			cold	sun	0.2
			cold	rain	0.3

$P(W T = \text{cold})$			$P(W T = \text{hot})$		
W	P		W	P	
sun	0.4		sun	0.8	
rain	0.6		rain	0.2	

Normalization Trick

$P(T, W)$			$P(W T = c)$	
T	W	P	W	P
hot	sun	0.4	sun	0.4
hot	rain	0.1	rain	0.6
cold	sun	0.2		
cold	rain	0.3		

Normalization Trick

$$P(W=s|T=c) = \frac{P(W=s, T=c)}{P(T=c)}$$

$$= \frac{P(W=s, T=c)}{P(W=s, T=c) + P(W=r, T=c)}$$

$$= \frac{0.2}{0.2+0.3} = 0.4$$

$P(T, W)$					
T	W	P			
hot	sun	0.4			
hot	rain	0.1			
cold	sun	0.2			
cold	rain	0.3			

SELECT the joint probabilities matching the evidence

$P(c, W)$					
T	W	P			
cold	sun	0.2			
cold	rain	0.3			

NORMALIZE the selection (make it sum to one)

$P(W T=c)$					
W	P				
sun	0.4				
rain	0.6				

$$P(W=r|T=c) = \frac{P(W=r, T=c)}{P(T=c)}$$

$$= \frac{P(W=r, T=c)}{P(W=s, T=c) + P(W=r, T=c)}$$

$$= \frac{0.3}{0.3+0.2} = 0.6$$

Normalization Trick

$P(T, W)$								
T	W	P						
hot	sun	0.4						
hot	rain	0.1						
cold	sun	0.2						
cold	rain	0.3						

SELECT the joint probabilities matching the evidence

$P(c, W)$								
T	W	P						
cold	sun	0.2						
cold	rain	0.3						

NORMALIZE the selection (make it sum to one)

$P(W T=c)$								
W	P							
sun	0.4							
rain	0.6							

- Why does this work? Sum of selection is P(evidence)! (P(T=c), here)

$$P(x_1|x_2) = \frac{P(x_1, x_2)}{P(x_2)} = \frac{P(x_1, x_2)}{\sum_{x_1} P(x_1, x_2)}$$

Quiz: Normalization Trick

- P(X | Y=-y) ?

$P(X, Y)$					
X	Y	P			
+x	+y	0.2			
+x	-y	0.3			
-x	+y	0.4			
-x	-y	0.1			

SELECT the joint probabilities matching the evidence

NORMALIZE the selection (make it sum to one)

To Normalize

- (Dictionary) To bring or restore to a normal condition

All entries sum to ONE

- Procedure:**
 - Step 1: Compute Z = sum over all entries
 - Step 2: Divide every entry by Z

- Example 1**

W	P		
sun	0.2	W	P
rain	0.3	sun	0.4
		rain	0.6

Normalize
Z = 0.5

- Example 2**

T	W	P
hot	sun	20
hot	rain	5
cold	sun	10
cold	rain	15

Normalize

Z = 50

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

Probabilistic Inference

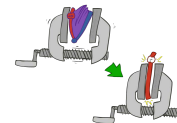
- Probabilistic inference: compute a desired probability from other known probabilities (e.g. conditional from joint)
- We generally compute conditional probabilities
 - P(on time | no reported accidents) = 0.90
 - These represent the agent's *beliefs* given the evidence
- Probabilities change with new evidence:
 - P(on time | no accidents, 5 a.m.) = 0.95
 - P(on time | no accidents, 5 a.m., raining) = 0.80
 - Observing new evidence causes *beliefs* to be updated



Inference by Enumeration

- General case:**
 - Evidence variables: $E_1 \dots E_k = e_1 \dots e_k$
 - Query* variable: Q
 - Hidden variables: $H_1 \dots H_r$
- We want:** $P(Q|e_1 \dots e_k)$
- Step 1: Select the entries consistent with the evidence**
- Step 2: Sum over H to get joint of Query and evidence**
- Step 3: Normalize**

X	P
-3	0.05
-1	0.25
1	0.2
5	0.01



$$P(Q, e_1 \dots e_k) = \sum_{h_1 \dots h_r} P(Q, h_1 \dots h_r, e_1 \dots e_k)$$

$$\times \frac{1}{Z}$$

$$Z = \sum_q P(Q, e_1 \dots e_k)$$

$$P(Q|e_1 \dots e_k) = \frac{1}{Z} P(Q, e_1 \dots e_k)$$

* Works fine with multiple query variables, too

Inference by Enumeration

- $P(W)$?
- $P(W \mid \text{winter})$?
- $P(W \mid \text{winter, hot})$?

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

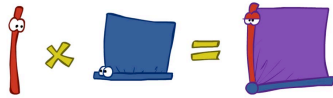
Inference by Enumeration

- Obvious problems:
 - Worst-case time complexity $O(d^n)$
 - Space complexity $O(d^n)$ to store the joint distribution

The Product Rule

- Sometimes have conditional distributions but want the joint


$$P(y)P(x|y) = P(x, y) \quad \Longleftrightarrow \quad P(x|y) = \frac{P(x, y)}{P(y)}$$



The Product Rule

$$P(y)P(x|y) = P(x, y)$$

- Example:

$P(W)$		$P(D W)$				$P(D, W)$		
R	P	D	W	P		D	W	P
wet	0.8	wet	sun	0.1		wet	sun	
sun	0.8	dry	sun	0.9		dry	sun	
rain	0.2	wet	rain	0.7		wet	rain	
		dry	rain	0.3		dry	rain	

The Chain Rule

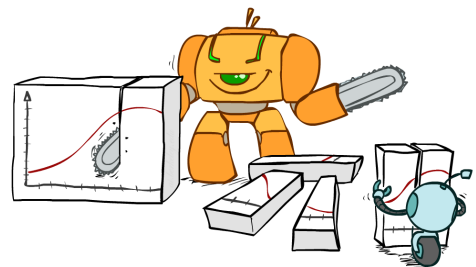
- More generally, can always write any joint distribution as an incremental product of conditional distributions

$$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

$$P(x_1, x_2, \dots, x_n) = \prod_i P(x_i|x_1 \dots x_{i-1})$$

- Why is this always true?

Bayes Rule



Bayes' Rule

- Two ways to factor a joint distribution over two variables:

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x)$$

That's my rule!

- Dividing, we get:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

- Why is this at all helpful?

- Lets us build one conditional from its reverse
- Often one conditional is tricky but the other one is simple
- Foundation of many systems we'll see later (e.g. ASR, MT)

- In the running for most important AI equation!



Inference with Bayes' Rule

- Example: Diagnostic probability from causal probability:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}$$

- Example:

- M: meningitis, S: stiff neck

$$\left. \begin{aligned} P(+m) &= 0.0001 \\ P(+s|+m) &= 0.8 \\ P(+s|-m) &= 0.01 \end{aligned} \right\} \text{Example gives}$$

$$P(+m|+s) = \frac{P(+s|+m)P(+m)}{P(+s)} = \frac{P(+s|+m)P(+m)}{P(+s|+m)P(+m) + P(+s|-m)P(-m)} = \frac{0.8 \times 0.0001}{0.8 \times 0.0001 + 0.01 \times 0.9999}$$

- Note: posterior probability of meningitis still very small
- Note: you should still get stiff necks checked out! Why?

Quiz: Bayes' Rule

- Given:

$P(W)$	
R	P
sun	0.8
rain	0.2

$P(D W)$		
D	W	P
wet	sun	0.1
dry	sun	0.9
wet	rain	0.7
dry	rain	0.3

- What is $P(W | \text{dry})$?

Ghostbusters, Revisited

- Let's say we have two distributions:

- Prior distribution over ghost location: $P(G)$
 - Let's say this is uniform
- Sensor reading model: $P(R | G)$
 - Given: we know what our sensors do
 - R = reading color measured at (1,1)
 - E.g. $P(R = \text{yellow} | G = (1,1)) = 0.1$

0.11	0.11	0.11
0.11	0.11	0.11
0.11	0.11	0.11

- We can calculate the posterior distribution $P(G|r)$ over ghost locations given a reading using Bayes' rule:

$$P(g|r) \propto P(r|g)P(g)$$

0.17	0.10	0.10
0.09	0.17	0.10
<0.01	0.09	0.17

[Demo: Ghostbuster – with probability (L12D2)]

Next Time: Markov Models